Ramdomness quality of CI chaotic generators. Application to Internet security

Jacques M. Bahi*, Xiaole Fang*, Christophe Guyeux* and Qianxue Wang*
*University of Franche-Comte
Computer Science Laboratory LIFC, Besançon, France
Email:jacques.bahi, xiaole.fang, christophe.guyeux, qianxue.wang@univ-fcomte.fr

arXiv:1112.0941v1 [cs.CR] 5 Dec 2011

***Abstract*—Due to the rapid development of the Internet in recent years, the need to find new tools to reinforce trust and security through the Internet has became a major concern. The discovery of new pseudo-random number generators with a strong level of security is thus becoming a hot topic, because numerous cryptosystems and data hiding schemes are directly dependent on the quality of these generators. At the conference Internet'09, we have described a generator based on chaotic iterations, which behaves chaotically as defined by Devaney. In this paper, the proposal is to improve the speed and the security of this generator, to make its use more relevant in the Internet security context. To do so, a comparative study between various generators is carried out and statistical results are given. Finally, an application in the information hiding framework is presented, to give an illustrative example of the use of such a generator in the Internet security field.**

***Keywords*-Internet security; Chaotic sequences; Statistical tests; Discrete chaotic iterations; Information hiding.**

I. Introduction

The development and popularity of the Internet, and its recent role in everyday life implies the need to protect data and privacy in digital world. This development has revealed new major security issues. For example, new concerns have recently appeared with the evolving of the Internet, as evoting, VoD or intellectual property protection. The pseudo-random number generators (PRNG) play an important role in all of these emerging techniques, because they are fundamental in cryptosystems and information hiding schemes. PRNGs are typically defined by a deterministic recurrent sequence in a finite state space, usually a finite field or ring, and an output function mapping each state to an input value. This is often either a real number in the interval $(0, 1)$ or an integer in some finite range [8]. Conventionally, PRNGs based on linear congruential methods and feedback shift-registers are popular [6].

To use a PRNG with a large level of security is necessary to satisfy the Internet security requirements recalled above. This level depends on the proof of theoretical properties and results of numerous statistical tests. Many PRNGs have been proven to be secure, following a probabilistic approach. However, recently, several researchers have been exploring the idea of using chaotic dynamical systems for this purpose [5] [3]. The random-like, unpredictable dynamics of chaotic systems, their inherent determinism and simplicity of realization suggest their potential for exploitation as PRNGs. Such generators can strongly improve security in information hiding and cryptography: due to unpredictability, the possibilities offered to an attacker to achieve his goal are drastically reduced. For example the keys of cryptosystems need to be unpredictable enough, making it impossible for any search optimization based on the reduction of the key space to the most probable values. But the number of generators claimed as chaotic, which actually have been proven to be unpredictable (as it is defined in the mathematical theory of chaos) is very small.

This paper extends a study initiated in [2] and [13], and tries to fill this gap. In [2], it is proven that chaotic iterations (CIs), a suitable tool for fast computing iterative algorithms, satisfies the topological chaotic property, as it is defined by Devaney [4]. In the paper [13] presented at Internet'09, the chaotic behavior of CIs is exploited in order to obtain an unpredictable PRNG, which depends on two logistic maps. We have shown that, in addition of being chaotic, this generator can pass the NIST (National Institute of Standards and Technology of the U.S. Government) battery of tests [11], widely considered as a comprehensive and stringent battery of tests for cryptographic applications. In this paper, we have improved the speed and security of the former generator. Chaotic properties, statistical tests and security analysis [14] allow us to consider that this generator has good pseudo-random characteristics and is capable to withstand attacks. Moreover, its high linear complexity and its large key space lead to the conviction that this generator is suitable for applications in the Internet security field. After having presented the theoretical framework of the study and a security analysis, we will give a comparison based on statistical tests. Finally a concrete example of how to use these pseudo-random numbers for information hiding through the Internet is detailed.

The rest of this paper is organized in the following way. In Section II, some basic definitions concerning chaotic iterations and PRNGs are recalled. Then, the generator based on discrete chaotic iterations is presented in Section III. Section IV is devoted to its security analysis. In Section V, various tests are passed with a goal to achieve a statistical comparison between this new PRNG and other existing ones. In Section VI, a potential use of this PRNG in some Internet security field is presented, namely in information hiding. The paper ends with a conclusion and intended future work.

II. Basic recalls

A. Notations

$[\![1; \mathsf{N}]\!]$	$\to \{1, 2, \ldots, N\}$
S^n	$\to$ the n^{th} term of a sequence $S = (S^1, S^2, \ldots)$
v_i	$\to$ the i^{th} component of a vector $v = (v_1, v_2, \ldots, v_n)$
f^k	$\to k^{th}$ composition of a function f
strategy	$\to$ a sequence which elements belong in $[\![1; \mathsf{N}]\!]$
$\mathbb{S}$	$\to$ the set of all strategies
$\mathbf{C}_n^k$	$\to$ the binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
$\oplus$	$\to$ bitwise exclusive or
$+$	$\to$ the integer addition
$\ll$ and $\gg$	$\to$ the usual shift operators
$(\mathcal{X}, \mathrm{d})$	$\to$ a metric space

$\lfloor x \rfloor \rightarrow$ returns the highest integer smaller than x
$n! \rightarrow$ the factorial $n! = n \times (n-1) \times \cdots \times 1$
$\mathbb{N}^* \rightarrow$ the set of positive integers {1,2,3,...}

B. Chaotic iterations

Definition 1 The set $\mathbb{B}$ denoting $\{0,1\}$, let $f : \mathbb{B}^{\mathsf{N}} \longrightarrow \mathbb{B}^{\mathsf{N}}$ be an "iteration" function and $S \in \mathbb{S}$ be a chaotic strategy. Then, the so-called *chaotic iterations* are defined by [12]

$$\begin{array}{l} x^0 \in \mathbb{B}^{\mathsf{N}}, \\ \forall n \in \mathbb{N}^*, \forall i \in [\![1;\mathsf{N}]\!], x_i^n = \left\{ \begin{array}{ll} x_i^{n-1} & \text{if } S^n \neq i \\ f(x^{n-1})_{S^n} & \text{if } S^n = i. \end{array} \right. \end{array} \tag{1}$$

In other words, at the n^{th} iteration, only the S^n-th cell is "iterated". Chaotic iterations generate a set of vectors (boolean vectors in this paper), which are defined by an initial state x^0, an iteration function f, and a chaotic strategy S.

C. XORshift

XORshift is a category of very fast PRNGs designed by George Marsaglia [9]. It repeatedly uses the transform of exclusive or (XOR) on a number with a bit shifted version of it. The state of a XORshift generator is a vector of bits. At each step, the next state is obtained by applying a given number of XORshift operations to w-bit blocks in the current state, where $w = 32$ or 64. A XORshift operation is defined as follows. Replace the w-bit block by a bitwise XOR of the original block, with a shifted copy of itself by a positions either to the right or to the left, where $0 < a < w$. This Algorithm 1 has a period of $2^{32} - 1 = 4.29 \times 10^9$.

Input: the internal state z (a 32-bits word)
Output: y (a 32-bits word)
$z \leftarrow z \oplus (z \ll 13)$;
$z \leftarrow z \oplus (z \gg 17)$;
$z \leftarrow z \oplus (z \ll 5)$;
$y \leftarrow z$;
return y;

Algorithm 1: An arbitrary round of XORshift algorithm

III. The new generation of CI pseudo-random sequence

A. Chaotic iterations as pseudo-random generator

1) Presentation: The novel generator is designed by the following process. First of all, some chaotic iterations have to be done to generate a sequence $(x^n)_{n\in\mathbb{N}} \in \left(\mathbb{B}^{\mathsf{N}}\right)^{\mathbb{N}}$ ($\mathsf{N} \in \mathbb{N}^*, \mathsf{N} \geqslant 2$, N is not necessarily equal to 32) of boolean vectors, which are the successive states of the iterated system. Some of these vectors will be randomly extracted and our pseudo-random bit flow will be constituted by their components. Such chaotic iterations are realized as follows. Initial state $x^0 \in \mathbb{B}^{\mathsf{N}}$ is a boolean vector taken as a seed (see Section III-A2) and chaotic strategy $(S^n)_{n\in\mathbb{N}} \in [\![1,\mathsf{N}]\!]^{\mathbb{N}}$ is an irregular decimation of a XORshift sequence (Section III-A4). The iterate function f is the vectorial boolean negation:

$$f_0 : (x_1, ..., x_{\mathsf{N}}) \in \mathbb{B}^{\mathsf{N}} \longmapsto (\overline{x_1}, ..., \overline{x_{\mathsf{N}}}) \in \mathbb{B}^{\mathsf{N}}.$$

At each iteration, only the S^i-th component of state X^n is updated, as follows: $x_i^n = x_i^{n-1}$ if $i \neq S^i$, else $x_i^n = \overline{x_i^{n-1}}$. Finally, some x^n are selected by a sequence m^n as the pseudo-random bit sequence of our generator. The sequence $(m^n)_{n\in\mathbb{N}} \in \mathcal{M}^{\mathbb{N}}$ is computed from a XORshift sequence $(y^n)_{n\in\mathbb{N}} \in [\![0, 2^{32}-1]\!]$ (see Section III-A3). So, the generator returns the following values:
Bits:

$$x_1^{m_0} x_2^{m_0} x_3^{m_0} \ldots x_{\mathsf{N}}^{m_0} x_1^{m_0+m_1} x_2^{m_0+m_1} \ldots x_{\mathsf{N}}^{m_0+m_1} x_1^{m_0+m_1+m_2} \ldots$$

or States:

$$x^{m_0} x^{m_0+m_1} x^{m_0+m_1+m_2} \ldots$$

2) The seed: The initial state of the system x^0 and the first term y^0 of the XORshift are seeded either by the current time in seconds since the Epoch, or by a number that the user inputs, as it is usually the case for every PRNG.

3) Sequence m of returned states: The output of the sequence (y^n) is uniform in $[\![0, 2^{32}-1]\!]$, because it is produced by a XORshift generator. However, we do not want the output of (m^n) to be uniform in $[\![0, N]\!]$, because in this case, the returns of our generator will not be uniform in $[\![0, 2^N - 1]\!]$, as it is illustrated in the following example. Let us suppose that $x^0 = (0, 0, 0)$. Then $m^0 \in [\![0, 3]\!]$.

- If $m^0 = 0$, then no bit will change between the first and the second output of our PRNG. Thus $x^1 = (0, 0, 0)$.
- If $m^0 = 1$, then exactly one bit will change, which leads to three possible values for x^1, namely $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$.
- *etc.*

As each value in $[\![0, 2^3 - 1]\!]$ must be returned with the same probability, then the values $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ must occur for x^1 with the same probability. Finally we see that, in this example, $m^0 = 1$ must be three times more probable than $m^0 = 0$. This leads to the following general definition for m:

$$m^n = f(y^n) = \left\{ \begin{array}{l} 0 \text{ if } 0 \leqslant \frac{y^n}{2^{32}} < \frac{C_N^0}{2^N}, \\ 1 \text{ if } \frac{C_N^0}{2^N} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{1} \frac{C_N^i}{2^N}, \\ 2 \text{ if } \sum_{i=0}^{1} \frac{C_N^i}{2^N} \leqslant \frac{y^n}{2^{32}} < \sum_{i=0}^{2} \frac{C_N^i}{2^N}, \\ \vdots \quad \vdots \\ N \text{ if } \sum_{i=0}^{N-1} \frac{C_N^i}{2^N} \leqslant \frac{y^n}{2^{32}} < 1. \end{array} \right. \tag{2}$$

4) Chaotic strategy: The chaotic strategy $(S^k) \in [\![1, N]\!]^{\mathbb{N}}$ is generated from a second XORshift sequence $(b^k) \in [\![1, N]\!]^{\mathbb{N}}$. The sole difference between the sequences S and b is that some terms of b are discarded, in such a way that: $\forall k \in \mathbb{N}, (S^{M^k}, S^{M^k+1}, \ldots, S^{M^{k+1}-1})$ does not contain a same integer twice, where $M^k = \sum_{i=0}^{k} m^i$. Therefore, no bit will change more than once between two successive outputs of our PRNG, increasing the speed of the former generator by doing so. S is said to be "an irregular decimation" of b. This decimation can be obtained by the following process.

Let $(d^1, d^2, \ldots, d^N) \in \{0,1\}^N$ be a mark sequence, such that whenever $\sum_{i=1}^{N} d^i = m^k$, then $\forall i, d_i = 0$ ($\forall k$, the sequence is reset when d contains m^k times the number 1). This mark sequence will control the XORshift sequence b as follows:

- if $d^{b^j} \neq 1$, then $S^k = b^j$, $d^{b^j} = 1$ and $k = k + 1$
- if $d^{b^j} = 1$, then b^j is discarded.

For example, if $b = 142\underline{2}33414\underline{2}11\underline{2}2\underline{3}4...$ and $m = 4241...$, then $S = 1423\ 34\ 1423\ 4...$ Another example is given in Table II-C, in which r means "reset" and the integers which are underlined in sequence b are discarded.

B. CI(XORshift, XORshift) algorithm

The basic design procedure of the novel generator is summed up in Algorithm 2. The internal state is x, the output state is r. a and b are those computed by the two XORshift generators. The value $f(a)$ is an integer, defined as in Equation 2. Lastly, N is a constant defined by the user.

Input: the internal state x (N bits)
Output: a state r of N bits
for $i = 0, \ldots, N$ **do**
  $d_i \leftarrow 0$;
end
$a \leftarrow XORshift1()$;
$m \leftarrow f(a)$;
$k \leftarrow m$;
for $i = 0, \ldots, k$ **do**
  $b \leftarrow XORshift2()\ mod\ \mathsf{N}$;
  $S \leftarrow b$;
  if $d_S = 0$ **then**
    $x_S \leftarrow \overline{x_S}$;
    $d_S \leftarrow 1$;
  end
  else if $d_S = 1$ **then**
    $k \leftarrow k + 1$;
  end
end
$r \leftarrow x$;
return r;

Algorithm 2: An arbitrary round of the new CI(XORshift,XORshift) generator

As a comparison, the basic design procedure of the old generator is recalled in Algorithm 3 (a and b are computed by Logistic maps, N and $c \geqslant 3\mathsf{N}$ are constants defined by the user). See [13] for further informations.

Input: the internal state x (N bits)
Output: a state r of N bits
$a \leftarrow Logisticmap1()$;
if $a > 0.5$ **then**
  $d \leftarrow 1$
end
else
  $d \leftarrow 0$
end
$m \leftarrow d + c$;
for $i = 0, \ldots, m$ **do**
  $b \leftarrow Logisticmap2()$;
  $S \leftarrow 100000b\ mod\ \mathsf{N}$;
  $x_S \leftarrow \overline{x_S}$;
end
$r \leftarrow x$;
return r;

Algorithm 3: An arbitrary round of the old PRNG

C. Illustrative example

In this example, $\mathsf{N} = 4$ is chosen for easy understanding. The initial state of the system x^0 can be seeded by the decimal part t of the current time. For example, if the current time in seconds since the Epoch is 1237632934.484088, so $t = 484088$, then $x^0 = t \pmod{16}$ in binary digits, *i.e.*, $x^0 = (0, 1, 0, 0)$.

To compute m sequence, Equation 3 can be adapted to this example as follows:

$$m^n = f(y^n) = \begin{cases} 0 & \text{if} \quad 0 \leqslant \frac{y^n}{2^{32}} < \frac{1}{16}, \\ 1 & \text{if} \quad \frac{1}{16} \leqslant \frac{y^n}{2^{32}} < \frac{5}{16}, \\ 2 & \text{if} \quad \frac{5}{16} \leqslant \frac{y^n}{2^{32}} < \frac{11}{16}, \\ 3 & \text{if} \quad \frac{11}{16} \leqslant \frac{y^n}{2^{32}} < \frac{15}{16}, \\ 4 & \text{if} \quad \frac{15}{16} \leqslant \frac{y^n}{2^{32}} < 1, \end{cases} \tag{3}$$

where y is generated by XORshift seeded with the current time. We can see that the probabilities of occurrences of $m = 0$, $m = 1$, $m = 2$, $m = 3$, $m = 4$, are $\frac{1}{16}$, $\frac{4}{16}$, $\frac{6}{16}$, $\frac{4}{16}$, $\frac{1}{16}$, respectively. This m determines what will be the next output x. For instance,

- If $m = 0$, the following x will be $(0, 1, 0, 0)$.
- If $m = 1$, the following x can be $(1, 1, 0, 0)$, $(0, 0, 0, 0)$, $(0, 1, 1, 0)$ or $(0, 1, 0, 1)$.
- If $m = 2$, the following x can be $(1, 0, 0, 0)$, $(1, 1, 1, 0)$, $(1, 1, 0, 1)$, $(0, 0, 1, 0)$, $(0, 0, 0, 1)$ or $(0, 1, 1, 1)$.
- If $m = 3$, the following x can be $(0, 0, 1, 1)$, $(1, 1, 1, 1)$, $(1, 0, 0, 1)$ or $(1, 0, 1, 0)$.
- If $m = 4$, the following x will be $(1, 0, 1, 1)$.

In this simulation, $m = 0, 4, 2, 2, 3, 4, 1, 1, 2, 3, 0, 1, 4, ...$ Additionally, b is computed with a XORshift generator too, but with another seed. We have found $b = 1, 4, 2, 2, 3, 3, 4, 1, 1, 4, 3, 2, 1, ...$

Chaotic iterations are made with initial state x^0, vectorial logical negation f_0 and strategy S. The result is presented in Table I. Let us recall that sequence m gives the states x^n to return, which are here $x^0, x^{0+4}, x^{0+4+2}, \ldots$ So, in this example, the output of the generator is: 10100111101111110011... or 4,4,11,8,1...

IV. Security analysis

A. Key space

The PRNG proposed in this paper is based on discrete chaotic iterations. It has an initial value $x^0 \in \mathbb{B}^{\mathsf{N}}$. Considering this set of initial values alone, the key space size is equal to 2^{N}. In addition, this new generator combines digits of two other PRNGs. We used two different XORshifts here. Let k be the key space of XORshift. So the total key space size is close to $2^{\mathsf{N}} \cdot k^2$. Lastly, the impact of Equation 2 must be taken into account. This leads to conclude that the key space size is large enough to withstand attacks.

B. Devaney's chaos property

Generally, the quality of a PRNG depends, to a large extent, on the following criteria: randomness, uniformity, independence, storage efficiency, and reproducibility. A chaotic sequence may satisfy these requirements and also other chaotic properties, as ergodicity, entropy, and expansivity. A chaotic sequence is extremely sensitive to the initial conditions. That is, even a minute difference in the initial state of the system can lead to enormous differences in the final state, even over fairly small timescales. Therefore, chaotic sequence fits the requirements of pseudo-random sequence well. Contrary to XORshift, our generator possesses these chaotic properties [2],[13]. However, despite a large number of papers published in the field of chaos-based pseudo-random generators,

m	0	4					2		2		
k	0	4			$+1$		2		2	$+1$	
b		1	4	2	$\underline{2}$	3	3	4	1	$\underline{1}$	4
d	r	r (1,0,0,0)	(1,0,0,1)	(1,1,0,1)		(1,1,1,1,)	r (0,0,1,0)	(0,0,1,1)	r (1,0,0,0)		(1,0,0,1)
S		1	4	2		3	3	4	1		4
x^0	x^0					x^4		x^6			x^8
0	0	$\xrightarrow{1}1$				1		1	$\xrightarrow{1}0$		0
1	1			$\xrightarrow{2}0$		0		0			0
0	0				$\xrightarrow{3}1$	1	$\xrightarrow{3}0$	0			0
0	0		$\xrightarrow{4}1$			1		$\xrightarrow{4}0$ 0		$\xrightarrow{4}1$	1

Binary Output: $x_1^0x_2^0x_3^0x_4^0x_1^0x_2^0x_3^0x_4^0x_1^4x_2^4x_3^4x_4^4x_1^6x_2^6... = 01000100101110000001...$
Integer Output: $x^0, x^0, x^4, x^6, x^8... = 4, 4, 11, 8, 1...$

Table I: Application example

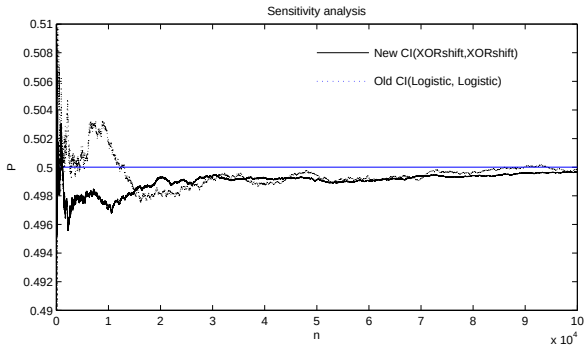


Figure 1: Sensitivity analysis

the impact of this research is rather marginal. This is due to the following reasons: almost all PRNG algorithms using chaos are based on dynamical systems defined on continuous sets (*e.g.*, the set of real numbers). So these generators are usually slow, requiring considerably more storage space and lose their chaotic properties during computations. These major problems restrict their use as generators [7].
In this paper we don't simply integrate chaotic maps hoping that the implemented algorithm remains chaotic. Indeed, the PRNG we conceive is just discrete chaotic iterations and we have proven in [2] that these iterations produce a topological chaos as defined by Devaney: they are regular, transitive, and sensitive to initial conditions. This famous definition of a chaotic behavior for a dynamical system implies unpredictability, mixture, sensitivity, and uniform repartition. Moreover, as only integers are manipulated in discrete chaotic iterations, the chaotic behavior of the system is preserved during computations, and these computations are fast.

C. Key sensitivity

As a consequence of its chaotic property, this PRNG is highly sensitive to the initial conditions. To illustrate this property, several initial values are put into the chaotic system. Let H be the number of differences between the sequences obtained in this way. Suppose n is the length of these sequences. Then the variance ratio P, defined by $P = H/n$, is computed. The results are shown in Figure 1 (x axis is sequence lengths, y axis is variance ratio P). For the two PRNGs, variance ratios approach 0.50, which indicates that the system is extremely sensitive to the initial conditions.

V. Statistical analysis

A. Basic usual tests

1) Comparative test parameters: In this section, five well-known statistical tests [10] are used as comparison tools. They encompass frequency and autocorrelation tests. In what follows, $s = s^0, s^1, s^2, \ldots, s^{n-1}$ denotes a binary sequence of length n. The question is to determine whether this sequence possesses some specific characteristics that a truly random sequence would be likely to exhibit. The tests are introduced in this subsection and results are given in the next one.

Frequency test (monobit test): The purpose of this test is to check if the numbers of 0's and 1's are approximately equal in s, as it would be expected for a random sequence. Let n_0, n_1 denote these numbers. The statistic used here is $X_1 = \frac{(n_0-n_1)^2}{n}$, which approximately follows a χ^2 distribution with one degree of freedom when $n \geqslant 10^7$.

Serial test (2-bit test): The purpose of this test is to determine if the number of occurrences of 00, 01, 10 and 11 as subsequences of s are approximately the same. Let n_{00}, n_{01}, n_{10}, and n_{11} denote the number of occurrences of $00, 01, 10$, and 11 respectively. Note that $n_{00} + n_{01} + n_{10} + n_{11} = n - 1$ since the subsequences are allowed to overlap. The statistic used here is:
$X_2 = \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 + n_1^2) + 1$, which approximately follows a χ^2 distribution with 2 degrees of freedom if $n \geqslant 21$.

Poker test: The poker test studies if each pattern of length m (without overlapping) appears the same number of times in s. Let $\lfloor \frac{n}{m} \rfloor \geqslant 5 \times 2^m$ and $k = \lfloor \frac{n}{m} \rfloor$. Divide the sequence s into k non-overlapping parts, each of length m. Let n_i be the number of occurrences of the i^{th} type of sequence of length m, where $1 \leqslant i \leqslant 2^m$. The statistic used is

$$X_3 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k,$$

which approximately follows a χ^2 distribution with $2^m - 1$ degrees of freedom. Note that the poker test is a generalization of the frequency test: setting $m = 1$ in the poker test yields the frequency test.

Runs test: The purpose of the runs test is to figure out whether the number of runs of various lengths in the sequence s is as expected, for a random sequence. A run is defined as a pattern of all zeros or all ones, a block is a run of ones, and a gap is a run of zeros. The expected number of gaps (or blocks) of length i in a random sequence of length n is

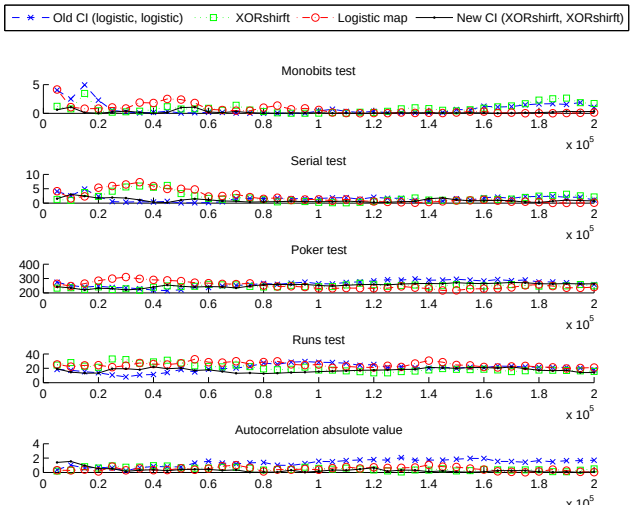


Figure 2: Comparison through various well-known tests

$e_i = \frac{n-i+3}{2^{i+2}}$. Let k be equal to the largest integer i such that $e_i \geqslant 5$. Let B_i, G_i be the number of blocks and gaps of length i in s, for each $i \in [\![1, k]\!]$. The statistic used here will then be:

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i},$$

which approximately follows a χ^2 distribution with $2k-2$ degrees of freedom.

Autocorrelation test: The purpose of this test is to check for coincidences between the sequence s and (non-cyclic) shifted versions of it. Let d be a fixed integer, $1 \leqslant d \leqslant \lfloor n/2 \rfloor$. The $A(d) = \sum_{i=0}^{n-d-1} s_i \oplus s_{i+d}$ is the amount of bits not equal between the sequence and itself displaced by d bits. The statistic used is: $X_5 = 2(A(d) - \frac{n-d}{2})/\sqrt{n-d}$, which approximately follows a normal distribution $N(0,1)$ if $n - d \geqslant 10$. Since small values of $A(d)$ are as unexpected as large values, a two-sided test should be used.

2) Comparison: We show in Table II a comparison between our new generator CI(XORshift, XORshift), its old version denoted Old CI(Logistic, Logistic), a PRNG based on logistic map, and a simple XORshift. Time (in seconds) is related to the duration needed by each algorithm to generate a 2×10^5 bits long sequence. The test has been conducted using the same computer and compiler with the same optimization settings for both algorithms, in order to make the test as fair as possible. Similar results have been achieved for different sequence lengths (see Figure 2). The results confirm that the proposed generator is a lot faster than the old one, while the statistical results are better for most of the parameters, leading to the conclusion that the new PRNG is more secure than the old one. Although the logistic map also has good results, it is too slow to be implemented in Internet applications.

B. NIST statistical test suite

Among the numerous standard tests for pseudo-randomness, a convincing way to prove the quality of the produced sequences is to confront them with the NIST (National Institute of Standards and Technology) Statistical Test Suite SP 800-22, released by the Information Technology Laboratory in August 25, 2008. This package of 15 tests was developed to measure the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic (pseudo-)random number generators. These tests focus on a variety of different types of non-randomness that could occur in such sequences.

Table III: SP 800-22 test results ($\mathbb{P}_T$)

Method	Old CI	New CI
Frequency (Monobit) Test	0.595549	0.474986
Frequency Test within a Block	0.554420	0.897763
Runs Test	0.455937	0.816537
Longest Run of Ones in a Block Test	0.016717	0.798139
Binary Matrix Rank Test	0.616305	0.262249
Discrete Fourier Transform (Spectral) Test	0.000190	0.007160
Non-overlapping Template Matching Test*	0.532252	0.449916
Overlapping Template Matching Test	0.334538	0.514124
Maurers Universal Statistical Test	0.032923	0.678686
Linear Complexity Test	0.401199	0.657933
Serial Test* (m=10)	0.013396	0.425346
Approximate Entropy Test (m=10)	0.137282	0.637119
Cumulative Sums (Cusum) Test*	0.046464	0.279680
Random Excursions Test*	0.503622	0.287409
Random Excursions Variant Test*	0.347772	0.486686
Success	15/15	15/15

(a) The original image

(b) The watermark

Figure 3: Original images

In our experiments, 100 sequences (s = 100) of 1,000,000 bits are generated and tested. If the value $\mathbb{P}_T$ of any test is smaller than 0.0001, the sequences are considered to not be good enough and the generator is unsuitable. Table III shows $\mathbb{P}_T$ of the sequences based on discrete chaotic iterations using different schemes. If there are at least two statistical values in a test, this test is marked with an asterisk and the average value is computed to characterize the statistical values. We can conclude from Table III that both the old generator and CI(XORshift, XORshift) have successfully passed the NIST statistical test suite.

VI. Application example in digital watermarking

In this section, an application example is given in the field of digital watermarking: a watermark is encrypted and embedded into a cover image using the scheme presented in [1] and CI(XORshift, XORshift). The carrier image is the well-known Lena, which is a 256 grayscale image, and the watermark is the 64×64 pixels binary image depicted in Figure 3.

The watermark is encrypted by using chaotic iterations: the initial state x^0 is the watermark, considered as a boolean vector, the iteration function is the vectorial logical negation, and

Table II: Comparison with Old CI(Logistic, Logistic) for a 2×10^5 bits sequence

Method	Monobit	Serial	Poker	Runs	Autocorrelation	Time
Logistic map	0.1280	0.1302	240.2893	26.5667	0.0373	0.965s
XORshift	1.7053	2.1466	248.9318	18.0087	-0.5009	0.096s
Old CI(Logistic, Logistic)	1.0765	1.0796	258.1069	20.9272	-1.6994	0.389s
New CI(XORshift,XORshift)	0.3328	0.7441	262.8173	16.7877	-0.0805	0.197s

(a) Differences with the original

(b) The encrypted watermark

Figure 4: Encrypted watermark and differences

the chaotic strategy $(S^k)_{k\in\mathbb{N}}$ is defined with CI(XORshift, XORshift), where initial parameters constitute the secret key and $N = 64$. Thus, the encrypted watermark is the last boolean vector generated by these chaotic iterations. An example of such an encryption is given in Figure 4.

Let L be the 256^3 booleans vector constituted by the three last bits of each pixel of Lena and U^k defined by:

$$\begin{cases} U^0 &= S^0 \\ U^{n+1} &= S^{n+1} + 2 \times U^n + n \ [mod\ 256^3] \end{cases} \quad (4)$$

The watermarked Lena I_w is obtained from the original Lena, whose three last bits are replaced by the result of 64^2 chaotic iterations with initial state L and strategy U (see Figure 4).

The extraction of the watermark can be obtained in the same way. Remark that the map $\theta \mapsto 2\theta$ of the torus, which is the famous dyadic transformation (a well-known example of topological chaos [4]), has been chosen to make $(U^k)_{k\leqslant 64^2}$ highly sensitive to the strategy. As a consequence, $(U^k)_{k\leqslant 64^2}$ is highly sensitive to the alteration of the image: any significant modification of the watermarked image will lead to a completely different extracted watermark, thus giving a way to authenticate media through the Internet.

VII. Conclusion and future work

In this paper, the pseudo-random generator proposed in [13] has been improved. By using XORshift instead of logistic map and due to a rewrite of the way to generate strategies, the generator based on chaotic iterations works faster and is more secure. The speed and randomness of this new PRNG has been compared to its former version, to XORshift, and to a generator based on logistic map. This comparison shows that CI(XORshift, XORshift) offers a sufficient speed and level of security for a whole range of Internet usages as cryptography and data hiding.

In future work, we will continue to try to improve the speed and security of this PRNG, by exploring new strategies and iteration functions. Its chaotic behavior will be deepened by using the various tools provided by the mathematical theory of chaos. New statistical tests will be used to compare this PRNG to existing ones. Additionally a probabilistic study of its security will be done. Lastly, new applications in computer science will be proposed, especially in the Internet security field.

References

[1] J. M. Bahi and C. Guyeux. A new chaos-based watermarking algorithm. In *SECRYPT 2010, International conference on security and cryptography*, pages ***–***, Athens, Greece, 2010. To appear.

[2] J. M. Bahi and C. Guyeux. Topological chaos and chaotic iterations, application to hash functions. *WCCI'10: 2010 IEEE World Congress on Computational Intelligence*, Accepted paper, 2010.

[3] S. Cecen, R. M. Demirer, and C. Bayrak. A new hybrid nonlinear congruential number generator based on higher functional power of logistic maps. *Chaos, Solitons and Fractals*, 42:847–853, 2009.

[4] R. L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Redwood City: Addison-Wesley, 2nd edition, 1989.

[5] M. Falcioni, L. Palatella, S. Pigolotti, and A. Vulpiani. Properties making a chaotic system a good pseudo random number generator. *arXiv*, nlin/0503035, 2005.

[6] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 1998.

[7] L. Kocarev. Chaos-based cryptography: a brief overview. *IEEE Circ Syst Mag*, 7:6–21, 2001.

[8] P. L'ecuyer. Comparison of point sets and sequences for quasi-monte carlo and for random number generation. *SETA 2008*, LNCS 5203:1–17, 2008.

[9] G. Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003.

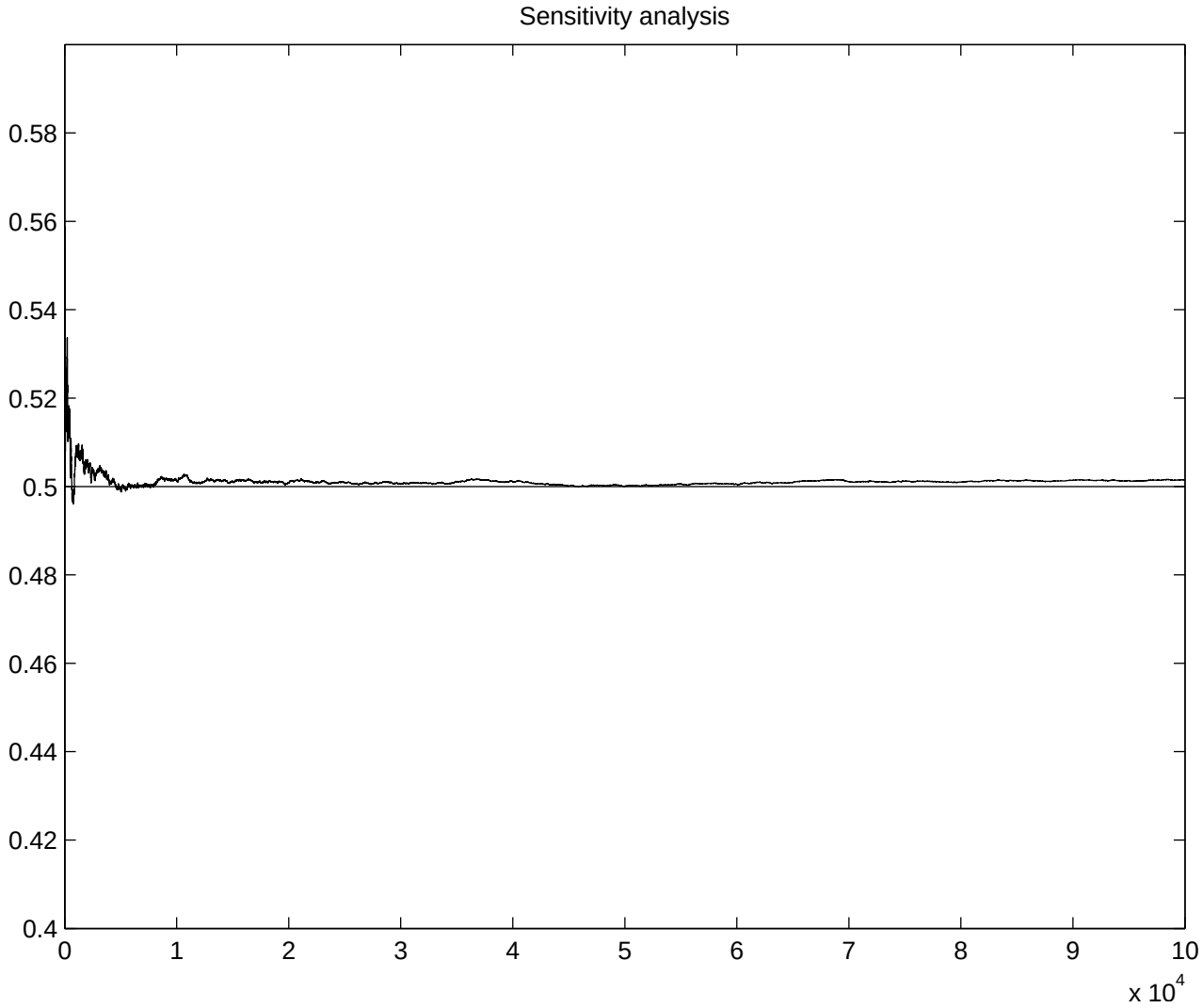
[10] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.

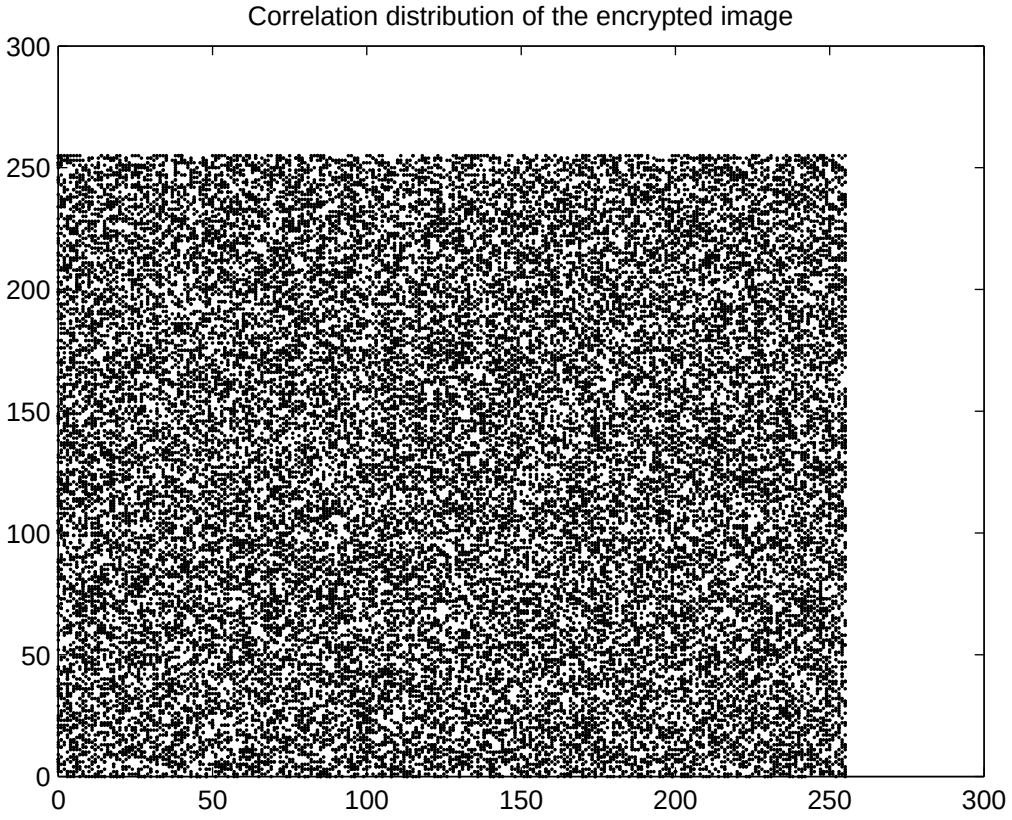
[11] NIST Special Publication 800-22 rev. 1. A statistical test suite for random and pseudorandom number generators for cryptographic applications. August 2008.

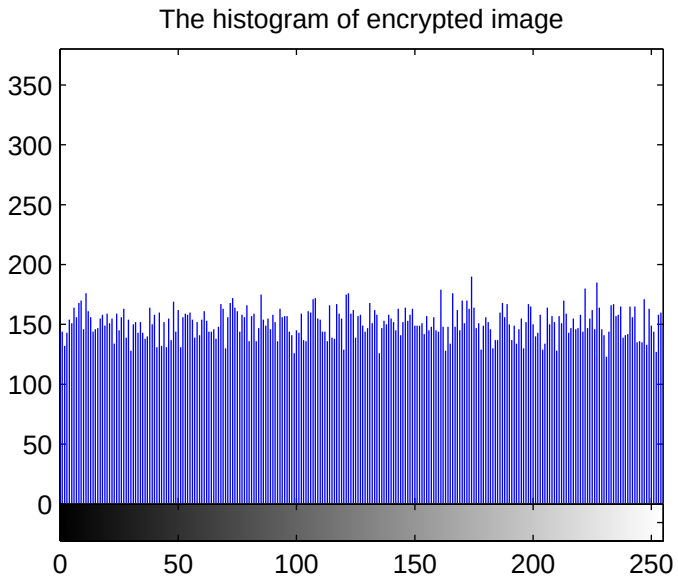
[12] F. Robert. *Discrete Iterations. A Metric Study*, volume 6. Springer Series in Computational Mathematics, 1986.

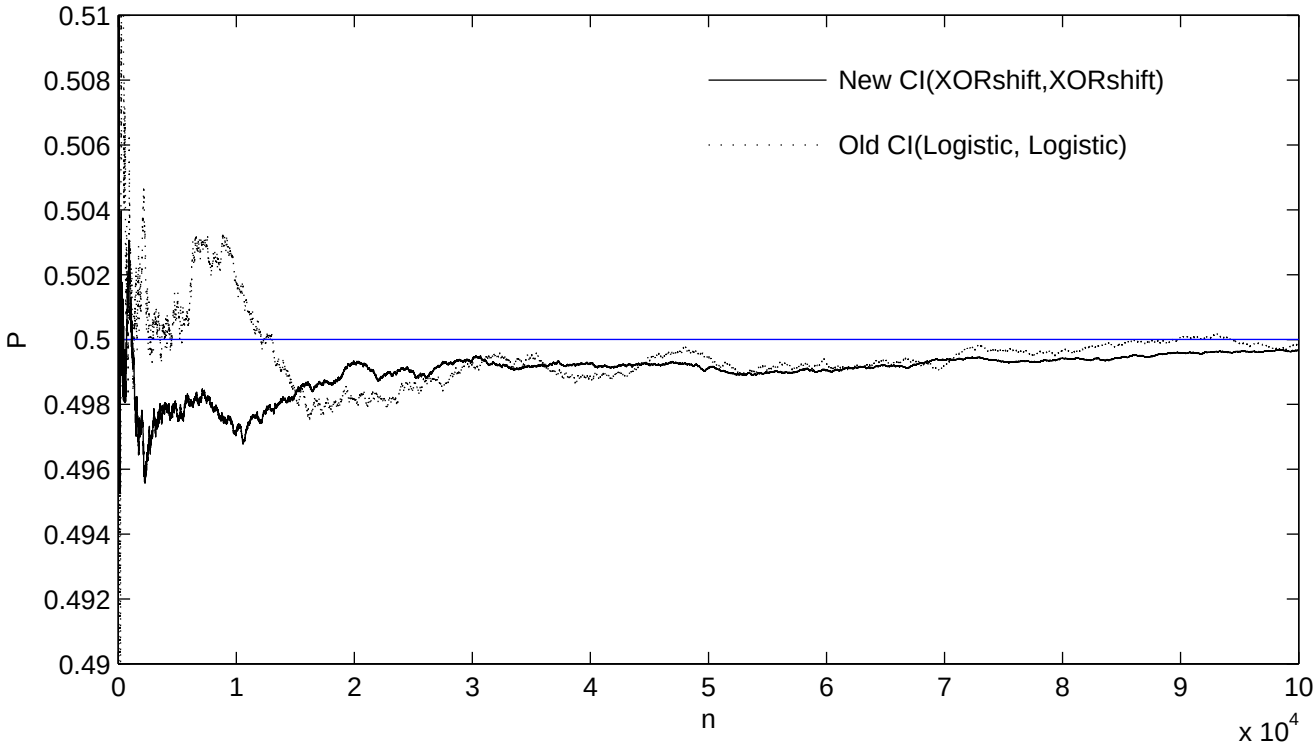
[13] Q. Wang, C. Guyeux, and J. M. Bahi. A novel pseudo-random generator based on discrete chaotic iterations for cryptographic applications. *INTERNET '09*, pages 71–76, 2009.

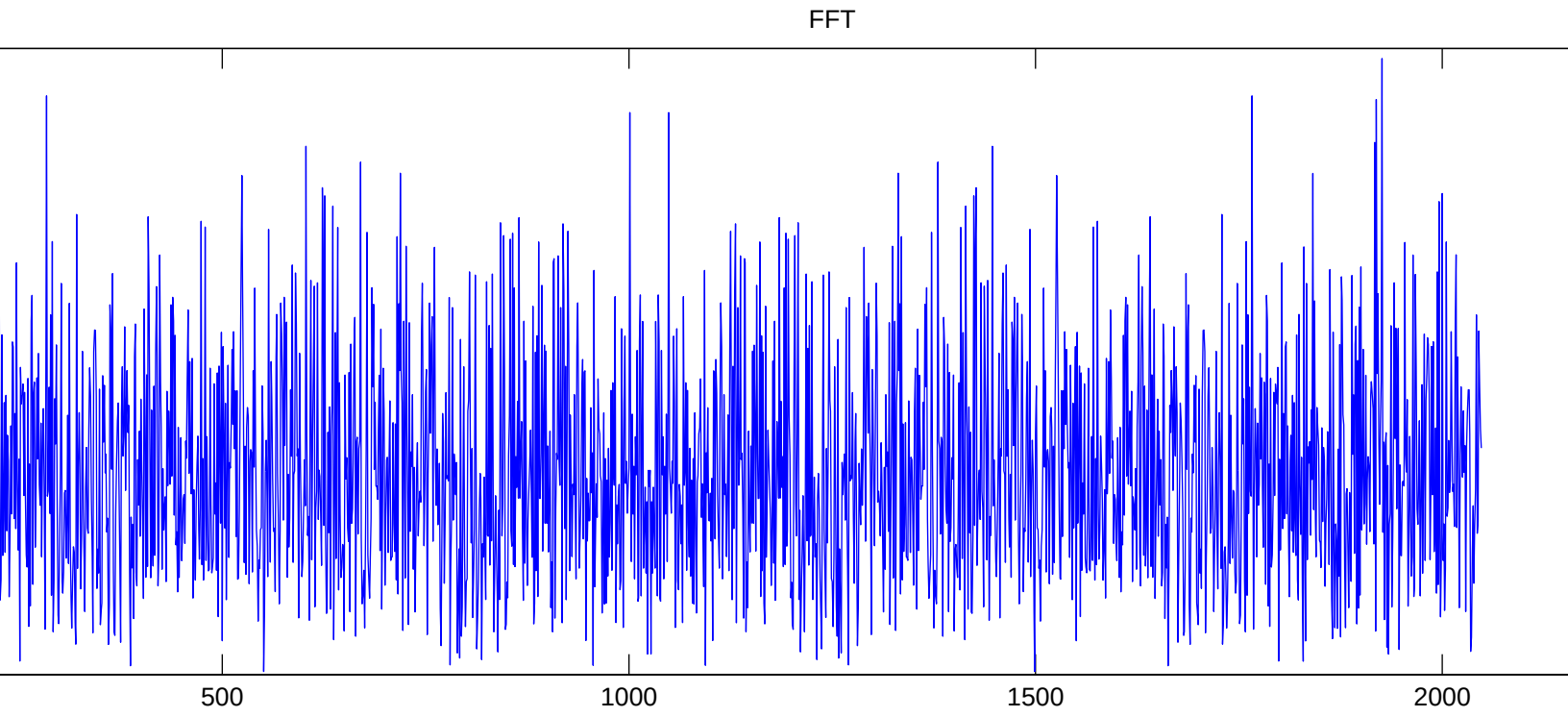
[14] F. Zheng, X. Tian, J. Song, and X. Li. Pseudo-random sequence generator based on the generalized henon map. *The Journal of China Universities of Posts and Telecommunications*, 15(3):64–68, 2008.

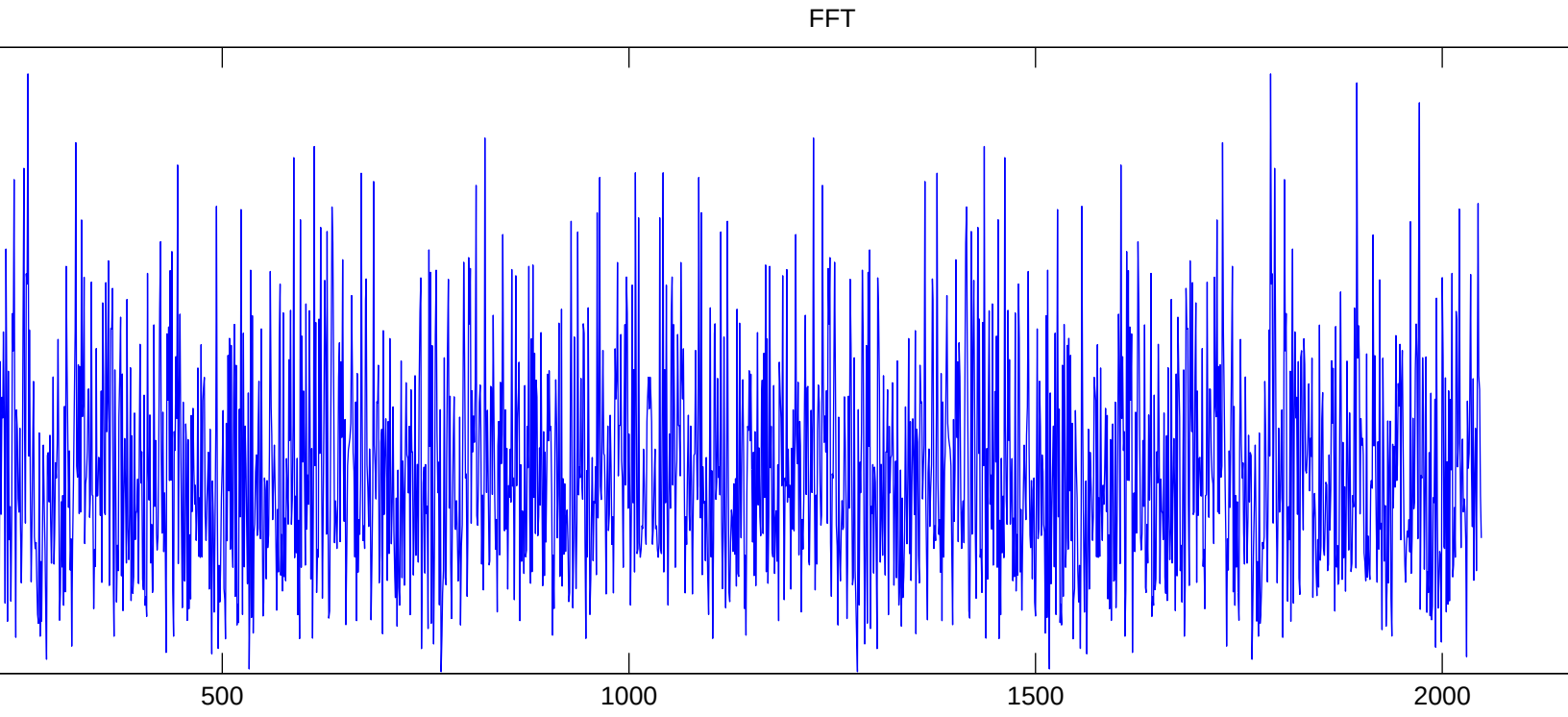

Sensitivity analysis
0.58
0.56
0.54
0.52
0.5
0.48
0.46
0.44
0.42
0.4
0
1
2
3
4
5
6
7
8
9
10
x 10^4


Correlation distribution of the encrypted image
300
250
200
150
100
50
0
0
50
100
150
200
250
300


The histogram of encrypted image
350
300
250
200
150
100
50
0
0
50
100
150
200
250


0.51
0.508
0.506
0.504
0.502
0.5
0.498
0.496
0.494
0.492
0.49
P
0
1
2
3
4
5
6
7
8
9
10
n
x 10^4
New CI(XORshift,XORshift)
Old CI(Logistic, Logistic)


FFT
500
1000
1500
2000


FFT
500
1000
1500
2000